Practice Questions for Quiz 1  

Scope: Chapters 1, 2, 3, 4, 5, and 7 in LYAH

Some standard built-in Haskell functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>Bool -&gt; Bool</td>
</tr>
<tr>
<td>(&amp;&amp;),(</td>
<td></td>
</tr>
<tr>
<td>(++)</td>
<td>[a] -&gt; [a] -&gt; [a]</td>
</tr>
<tr>
<td>(:)</td>
<td>a -&gt; [a] -&gt; [a]</td>
</tr>
<tr>
<td>concat</td>
<td>[[a]] -&gt; [a]</td>
</tr>
<tr>
<td>concatMap</td>
<td>(a -&gt; [b]) -&gt; [a] -&gt; [b]</td>
</tr>
<tr>
<td>drop, take</td>
<td>Int -&gt; [a] -&gt; [a]</td>
</tr>
<tr>
<td>elem</td>
<td>(Eq a) =&gt; a -&gt; [a] -&gt; Bool</td>
</tr>
<tr>
<td>filter</td>
<td>(a -&gt; Bool) -&gt; [a] -&gt; [a]</td>
</tr>
<tr>
<td>foldl</td>
<td>(b -&gt; a -&gt; b) -&gt; b -&gt; [a] -&gt; b</td>
</tr>
<tr>
<td>foldr</td>
<td>(a -&gt; b -&gt; b) -&gt; b -&gt; [a] -&gt; b</td>
</tr>
<tr>
<td>fst</td>
<td>(a,b) -&gt; a</td>
</tr>
<tr>
<td>snd</td>
<td>(a,b) -&gt; b</td>
</tr>
<tr>
<td>head, last</td>
<td>[a] -&gt; a</td>
</tr>
<tr>
<td>init, tail, reverse</td>
<td>[a] -&gt; [a]</td>
</tr>
<tr>
<td>length</td>
<td>[a] -&gt; Int</td>
</tr>
<tr>
<td>map</td>
<td>(a -&gt; b) -&gt; [a] -&gt; [b]</td>
</tr>
<tr>
<td>max, min</td>
<td>(Ord a) =&gt; a -&gt; a -&gt; a</td>
</tr>
<tr>
<td>maximum, minimum</td>
<td>(Ord a) =&gt; [a] -&gt; a</td>
</tr>
<tr>
<td>null</td>
<td>a -&gt; [a] -&gt; Bool</td>
</tr>
<tr>
<td>splitAt</td>
<td>Int -&gt; [a] -&gt; ([a],[a])</td>
</tr>
<tr>
<td>sum, product</td>
<td>(Num a) =&gt; [a] -&gt; a</td>
</tr>
<tr>
<td>zip</td>
<td>[a] -&gt; [b] -&gt; [(a,b)]</td>
</tr>
<tr>
<td>zipWith</td>
<td>(a -&gt; b -&gt; c) -&gt; [a] -&gt; [b] -&gt; [c]</td>
</tr>
</tbody>
</table>

Question 1. Write a Haskell function

longest :: [[[a]]] -> [a]

such that, given xss, a nonempty list of lists, (longest xss) returns a longest list in xss. Example: (longest [[2,4],[2,4,8,3],[[]]]) ~ [2,4,8,3].

Question 2. Write a function

assembleBy :: (a -> Bool) -> [a] -> ([[a]],[a])

such that if (assembleBy test xs) ~ (passes,fails), then passes is a list of all those elements of xs on which test returns True and fails is a list of all the elements of xs on which test returns False. Note that (assembleBy test []) should return ([],[]). Example:

assembleBy (<10) [[1,23,9,12,13],[1,9,0],[23,12,13]] ~ ([1,9,0],[23,12,13])

Question 3. This problem involves the following data-type for multiway trees:

```haskell
data Multi = Fork Int [Multi] deriving (Eq,Show)
```

**Definition.** (Fork n ts) is a leaf node iff ts=[]; and when ts≠[], (Fork n ts) is an interior node.

Write a function

```haskell
collectLeaves :: Multi -> [Int]
```
such that (collectLeaves t) returns a list of all the Int-labels of the leaf nodes of t. Examples:

```haskell
collectLeaves (Fork 3 [[]])  ~ [3]
collectLeaves (Fork 3 [Fork 10 [], Fork 400 []])  ~ [10,400]
```

Question 4. This problem involves the following data-type for binary trees:

```haskell
data BinTree = Emp | Branch Int BinTree BinTree deriving (Eq,Show)
```

Write a function

```haskell
numPath :: BinTree -> Int -> [Int]
```
such that (numPath t k) returns [] if k is not in t; but if k is in t, then (numPath t k) returns a list of numbers on a path from the root Branch-node to a Branch-node with value k. (Hint: Local variables (via let or where can be very useful here. Also (not (null xs)) tests if the list xs is nonempty.) Examples:

```haskell
(numPath t 6) ~ []
(numPath t 3) ~ [10,32,3]
```

For t =

```
    10
   /|
  /  |
 32  9
 /  /|
7  3 32  0
```

Page 1 of 2 8 February 2018
Practice Questions for Quiz 1

An answer to question 1

\[
\text{longest} :: [[a]] \to [a] \\
\text{longest} [xs] = xs \\
\text{longest} (xs:xss) = \text{if} \ (\text{length} \ xs) > (\text{length} \ ys) \ \text{then} \ xs \ \text{else} \ ys \\
\text{where} \ ys = \text{longest} \ xss
\]

Possible answers to question 2

\[
\text{assembleBy} \ p \ [] = ([],[])
\]
\[
\text{assembleBy} \ p \ (x:xs) = \text{if} \ p \ x \ \text{then} \ (x:ys,zs) \ \text{else} \ (ys,x:zs) \\
\text{where} \ (ys,zs) = \text{assembleBy} \ p \ xs
\]

\[
\text{assembleBy'} \ p \ xs = ([x \mid x<-xs, p x],[x \mid x<-xs, \text{not}(p \ x)])
\]

\[
\text{assembleBy''} \ p \ xs = (\text{filter} \ p \ xs, \text{filter} \ (\text{not} . p) \ xs)
\]

Possible answers to question 3

\[
\text{collectLeaves, collectLeaves'} :: \text{Multi} \to [\text{Int}]
\]
\[
\text{collectLeaves} \ (\text{Fork} \ k \ []) = [k]
\]
\[
\text{collectLeaves} \ (\text{Fork} \ _ \ ts) = \text{concatMap} \ \text{collectLeaves} \ ts
\]

\[
\text{collectLeaves'} \ (\text{Fork} \ k \ []) = [k]
\]
\[
\text{collectLeaves'} \ (\text{Fork} \ _ \ ts) = \text{concat} \ [\text{collectLeaves} \ t \mid t <- ts]
\]

\[
\text{collectLeaves''} \ (\text{Fork} \ k \ ts)
\]
\[
= [k \mid \text{null} \ ts] ++ \text{concatMap} \ \text{collectLeaves''} \ ts
\]

Possible answers to question 4

\[
\text{numPath, numPath'} :: \text{BinTree} \to \text{Int} \to [\text{Int}]
\]
\[
\text{numPath} \ (\text{Branch} \ m \ tl \ tr) \ k
\]
\[
| \ m==k = [m] \\
| \ \text{not} (\text{null} \ pl) = m:pl \\
| \ \text{not} (\text{null} \ pr) = m:pr \\
| \ \text{otherwise} = []
\]
\[
\text{where}
\]
\[
pl = \text{numPath} \ tl \ k \\
pr = \text{numPath} \ tr \ k
\]

\[
\text{numPath'} \ (\text{Branch} \ m \ tl \ tr) \ k
\]
\[
= \text{if} \ m==k \\
\text{then} \ [m] \\
\text{else} \ \text{head} \ ([m:ns \mid ns<- [\text{numPath'} \ tl \ k, \text{numPath'} \ tr \ k], \\
\text{not} (\text{null} \ ns)]++)
\]